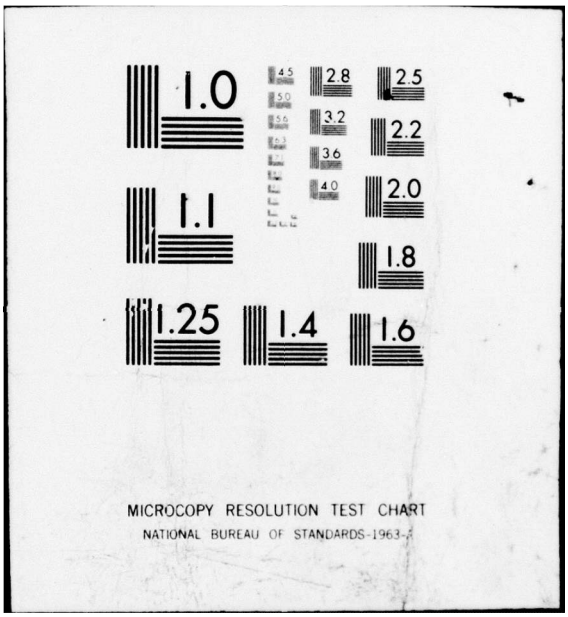
AD-A039 744 FEDERAL COBOL COMPILER TESTING SERVICE WASHINGTON D C F/G 9/2
AN EXPERIMENT IN THE USE OF SYNTHETIC PROGRAMS FOR SYSTEM BENCH--ETC(U)
MAY 77 P OLIVER, G N BAIRD, M M COOK

UNCLASSIFIED FCCTS/TR-77/07 NL

| OF |
AD
A039 744

END

DATE
FILMED
6-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

An experiment in the use of synthetic programs for system benchmarking

by PAUL OLIVER, GEORGE BAIRD, MARGARET COOK, ARNOLD JOHNSON and PATRICK HOYT

Department of the Navy
Washington, D.C.

BACKGROUND

Competitive computer system selection requires a tool for minimum performance measurement. The selection process must be fair and, ideally, brief and economical. Thus, the measurement tool must be visibly fair and impartial in its measurement of a computer system, it must relate what is being measured to user needs, and it must be economical to apply. The thrust of several ongoing "standard benchmark" efforts in the Department of Defense and other Federal Government agencies is to develop a measurement tool with these qualities.

There are several characteristics of computer systems which can be measured for the purpose of selection:

(a) Availability of equipment and software, in terms of reliability, maintenance time, and the like.

(b) Work capacity, which can be measured from a variety of viewpoints. *Job time* is a single-job measure and, therefore, not often used. *System throughput* is a measure of how much work is done, and is a function of the job mix and job load, as well as various system parameters. *Response time* is a measure of the quality of service rendered, and is largely dependent on operating system and hardware characteristics.

(c) Functional capabilities are susceptible to qualitative judgments, but demonstrations of these capabilities are often required of computer system vendors (e.g., a demonstration of an on-line text editor).

In the context of computer selection, we have felt it prudent to limit the scope of our efforts to measuring throughput capacity, recognizing, however, that the other factors may take on paramount importance under varying circumstances.

Relation to performance evaluation

It is important that we recognize the affinity of any benchmark study to the subject of computer performance evaluation, since some combination of evaluation techniques will of necessity be used in the development of "standard benchmarks." These techniques can be broadly classified and characterized as follows:[1]

(a) *Task-oriented* techniques concern themselves with system throughput capabilities with respect to a given workload. Simple instruction timings reduce the "workload" to specific classes of instructions (add time, floating-point multiply, etc.). Instruction mixes consist of "representative" samples of instruction sets designed to reflect the degree to which each instruction class is used for a given type of application. These are adequate for estimating *processor* power, but completely ignore memory, degree of multiprogramming, I/O loads, etc. Kernels are relatively small sequences of code performing a single (simple) function (e.g., a table search), and, again, are designed primarily for measuring processing power. The timings for kernels may be obtained by actually executing them or by hand-calculations. Benchmarks consist of a subset of a given workload ("natural" benchmarks), a subset which has been further modified ("hybrid" benchmarks), or a set of programs written specifically for the purpose of making a comparative evaluation ("synthetic" programs). Benchmarks are processed on the configurations being evaluated or compared, and the processing time is used as a relative figure of merit.

(b) The emphasis in *component-oriented* evaluation techniques is on the system being evaluated rather than on the workload to be processed by this system. Hardware monitors are relatively inexpensive, precise in what they measure, non-disruptive, but insensitive to data-dependent information. The characteristics of software monitors are almost the precise opposite of those for hardware monitors. The convenience of queueing models is offset by their inaccuracy and shallowness. Stochastic models (simulation models) are less imprecise but costly, and suffer from a credibility gap.

Problems with natural or hybrid benchmarks

Benchmarks have for some period of time constituted the accepted form of minimum performance measurement in computer selection throughout the Federal marketplace. Natural or hybrid benchmarks have the advantages of dealing

```
                X-30    PRINTER
                ...
   INPUTS       X-65    UNIVAC-1108
                X-66    UNIVAC-1108
                 ...

                SOURCE COMPUTER.
                  XXXXX65.
POPULATION      OBJECT COMPUTER.
FILE              XXXXX66.
FORMAT
                ...
                FILE CONTROL.  SELECT RESULTS ASSIGN TO
                  XXXXX30.
                ...
                SOURCE COMPUTER.
COMPILATION       UNIVAC-1108.
TIME
FORMAT          OBJECT COMPUTER.
                  UNIVAC-1108.
                ...
                FILE CONTROL.  SELECT RESULTS ASSIGN TO PRINTER.
```

Figure 1—Example of VP-Routine input, population file form of audit routines, and compilation-time form of audit routines

with a real system (thus avoiding half of the simulation credibility problem) and a "semi-real" job mix. Among the more serious problems associated with benchmarks are the following:

(a) It is extremely difficult, except in the simplest situations, to construct a set of benchmark programs which accurately reflects a given job mix. This of course is a problem common to any performance measurement technique, since the nature of "a given job mix" is dependent on a multitude of parameters, many of which are system dependent (e.g., *EX*ecute *C*hannel *P*rogram instruction counts are often used to measure I/O time on IBM S/360 or S/370 systems, but these instructions have little meaning outside the S/360-370 series, and often have no precise counterparts on other systems) and most of which are time dependent.

(b) They are generally non-portable (system dependent) and often do not run correctly, even on their native system.

(c) They are prepared and processed using a variety of procedures resulting in unduly long execution times, unreasonable file volumes, and inconsistent measurement procedures. This author has seen benchmarks for which the required processing time was better than three hours, and the file population resided on two dozen (full) tape reels! In some cases only processor time is measured; in others, all components (including, e.g., printers) must halt before timing stops.

(d) The above problems result in extremely high costs, to buyers and vendors, in terms of both time and money. It is not unusual for a vendor to spend 6-9 calendar months just to prepare the submitted benchmarks for processing, or for the cost of processing them to be 10 percent or more of the eventual bid price.

SCOPE OF THE U. S. NAVY EXPERIMENT

The Software Development Division of the Department of the Navy Automatic Data Processing Equipment Selection Office (ADPESO) is performing an experiment to determine the suitability of synthetic programs in alleviating the problems created by natural and hybrid benchmarks.

The experiment began in June 1973, with the development of a small (5 program) reference library of synthetic programs. We assumed that synthetic programs could be written so that relatively few parameters control their behavior; experimentation could be performed on these programs so that their behavior relative to changing parameter values would be predictable; specifications of a workload *based on the parameters implicitly defined by the synthetic programs* could be made, and synthetic program parameters could be set so as to reflect this workload.

The use of synthetic programs in performance evaluation does not represent a new concept. Dopping,[2] and Gosden and Sisson[3] reported on experiments in the use of synthetic programs as far back as 1962. More recent suggestions on their use have come from Joslin[4] and Buchholtz.[5] Our aims have been to obtain quantitative profiles of certain synthetic programs and to determine the scope of their feasible utility.

RELATED EFFORTS

There are several complementary efforts in the Federal Government aimed at designing representative benchmarks.

The U. S. Army Computer System Support and Evaluation Command has recently issued a solicitation for a "Standard Benchmark Study." The contract objectives are (a) The definition of all tasks and measurable functions performed by a computer in executing business-type applications; (b) Development of a method or technique of identifying and measuring the occurrence of each function or parameter in

```
PROJECT:  SYNTHETIC BENCHMARKS

MODULE:  SEQUENTIAL I/O

COMPILE TIME PARAMETERS:

    1.  Records/Block - for all files; impacts buffering.

    2.  Record Size - for all files and to reflect application.

    3.  Start Variable - used to vary accuracy requirement in compute
        kernel.

    4.  Table Size - to impact memory requirements.

    5.  Data Types - to reflect application.

EXECUTE TIME PARAMETERS:

    1.  Master File Size - to impact i/o time

    2.  Detail File Size - in conjunction with "repetitions" can impact
        processing time.

    3.  Repetitions - number of repetitions of a compute kernel per
        master-detail match.


NOTE:  See listing for more details.
```

Figure 2—Sequential I/O module parameters

each task for the purpose of profiling computer workloads. This solicitation is the result of a careful study on the part of a Department of Defense Joint Steering Committee which has, among other things, defined a preliminary set of application tasks and task parameters for benchmark purposes.

The Department of Agriculture has constructed a comprehensive set of benchmark programs which include transaction processing and data base management applications. There is much in this package which should be carefully studied as part of any effort at designing a library of standard benchmark programs.

The Department of Labor is developing a job selection simulation model[6] using actual utilization statistics as control parameters. Although the goals here are somewhat different from those of the "standard benchmark effort" there may be some related spinoff benefits.

A similar project is being carred on by Marine Corps using hardware monitors to provide data for the synthetic creation of jobs.[7]

RESULTS

The programs

Five processing tasks were selected as representing, in varying combinations, a broad variety of application tasks. These were sequential file processing, indexed sequential file processing, relative I/O processing, sorting, and computation.

Programs were written to perform each of these tasks. Because most of the Navy's present benchmark needs relate

PROJECT: SYNTHETIC BENCHMARKS

MODULE: INDEXED SEQUENTIAL UPDATE

COMPILE TIME PARAMETERS:

1. Memory Variable - is set by adjusting the size of a table in working-storage. This is available to vary the memory storage requirement of the program.
2. Record size - Default is 800 characters.
3. Block size - Default is 10.
4. Index key size - Default is 10.

EXECUTE TIME PARAMETERS

1. Master File Size - sets the number of records to be created for the master file.
2. Detail File Size - sets the number of transactions to be processed against the master file to measure I-O processing.
   (a) Deletion Percent - is percent of detail transactions which initiate deletion of master records (default is 10 percent). This parameter is available to measure the affect of record deletion type transactions on I-O processing time.
   (b) Addition Percent - is percent of detail transactions which add records to the master file (default is 10 percent). This parameter is available to measure the affect of transaction insertion into the index file on I-O processing time.
   (c) Sequential Percent - percent of detail transaction which initiate processing the index file sequentially (default is 5 percent). This is to measure the affect on I-O processing when accessing the index file sequentially.
3. Computation Repetitions - sets the number of times the program cycles through compute bound procedures. This parameter is available to place a workload on the CPU.

Figure 3—ISAM module parameters

PROJECT: SYNTHETIC BENCHMARKS

MODULE: RELATIVE I/O

COMPILE TIME PARAMETERS:

1. Master and Detail Files Record Size - minimum of 120 characters - the user can request a larger record.
2. Master and Detail Files Block Size - minimum of 1 record per block - the user may request a larger block size if applicable.

EXECUTE TIME PARAMETERS:

1. Number of Master Records -(5,000 default) - the user could request a larger or smaller number of records.
2. Order the Records are created - (sequential default) - The default causes the files to be created with 10% "missing" records, i.e. 5,000 records from 1 to 5,500. The user may request that a different percentage of gaps be left between records for inserting purposes.
3. Number of Detail Records - (2,500 default)
4. Percent of the Following - (100% total):
   (a) Detail records which match master records and cause an update to take place. (33% default).
   (b) Detail records which match master records and cause the master record to be deleted.
   (c) Detail records which do not match master records and cause a new master record to be created

Figure 4—Relative I/O module parameters

to COBOL-oriented workloads, all of the reference library programs are written in American National Standard COBOL. Additionally, all the programs are in "system independent" form. This is accomplished through the use of an executive program, the *VP-Routine*. The VP-Routine was developed in 1969 by the Department of the Navy as part of its *COBOL Compiler Validation System*.[8] It is used to resolve implementor names (e.g., in the ENVIRONMENT DIVISION), modify compile-time parameters (e.g., record sizes, precision, blocking factors), and automatically generate job control instructions appropriate to the system we are executing under (Figure 1).

Each program is controlled by a set of compile time and execution time parameters. Figures 2-6 identify these for each of the five programs. The ability to vary automatically certain parameters at compile time provides us with the flexibility to develop a fairly rich mix from just a few basic programs.

We have adopted certain design principles which, while applicable to software design in general, we felt were particularly important to this project.

(a) We have attempted to make every detail of the structure of each program visible and understandable to a prospective user. This is a prerequisite to a "sellable" product.

(b) The design of each program is consistent with that of the others. We have used "modular programming" throughout, although, frankly, this was simply a reflection of following long accepted standards of good programming practice. We maintained consistency in the binding time of parameters across programs. Thus, if a given parameter is bound at compile time in one program it is bound at compile time in all the programs. Also, all files used by a program are generated by that program (eventually, the file generation modules may be combined into one program).

(c) We have isolated the function of each of the program parameters so as to render each parameter independent of

PROJECT: SYNTHETIC BENCHMARKS

MODULE: SORT

COMPILE TIME PARAMETERS:

1. Record Length - Used to impact
   (a) Buffer size.
   (b) Transfer time.
   (c) Internal and external storage requirements
   (d) Whether minimum and maximum logical size of applications can be handled.
   (e) Whether sort can handle variable length logical records.
2. Blocking Factors - Used to affect
   (a) Buffer size.
   (b) Transfer time.
   (c) Ratio of inter-record gaps/data for magnetic tape; hence external storage requirements.
   (d) Mass storage partition use/waste ration; hence mass storage requirements and number of seeks and transfers required.
   (e) Whether minimum and maximum physical record size can be handled.
   (f) Whether padding is required.
   (g) Whether extra characters must be added to each physical record if the file is blocked.
   (h) Provides a way to increase I/O time used for a single transfer to change I/O to computer ratio.
3. Number of Sort Keys - Affects
   (a) Number of sort passes required to produce specified sequence.
   (b) Test that the number of keys allowed in a single sort step equals those required by an application.
   (c) Total length of sort field.
4. Type of Sort Keys - Determines
   (a) Whether all types of keys required by an application can be handled (numeric, alphabetic, alphanumeric, signed, decimal points).
   (b) Time required for various types of comparisons, numeric vs. alphanumeric.
   (c) Points out the collating sequence used by the machine for sorts and compares.
5. Order of Sort Keys - Prevent cheating by setting at test time to compare results against predicted behavior of final sort sequence.
6. Ascending or descending sort -

EXECUTE TIME PARAMETERS:

1. Number of Records -
   (a) Total data volume for input.
   (b) Whether sort can be done completely in core.
   (c) Amount of dependence on mass storage for intermediate merge strings.
2. Number of Computations on I/O -
   (a) Ability to simulate amount of modification done during sort process.
   (b) Changes ratio of added computer requirements/sort I/O processing.

NOTE: Although not specifically specified as a compile time parameter, the file assignments for INPUT-FILE, SORT-FILE, and OUTPUT-FILE can change the basic sort characteristics from mass storage to tape orientation. This affects file rewind time, transfer rates, and blocking conventions.

Figure 5—SORT module parameters

the others. This was necessary to avoid facing an exponentially rising set of options in setting parameters to control program behavior. This was a difficult principle to follow since, for example, a simple specification such as how one is to control I/O time can be made in terms of file size, blocking factor, logical record size, etc. In this case we could choose to use file size to effect time, blocking factor to impact buffering, and maintain logical record size constant.

(d) Only those functions which were felt essential to the accurate modeling of a task were included in each program. Thus we opted for a clearly defined scope and simplicity rather than complexity. We feel this was particularly important in the selection of synthetic program functions and parameters, since a lack of frugality can lead to a level of

PROJECT: SYNTHETIC BENCHMARKS

MODULE: COMPUTE

COMPILE TIME PARAMETERS:

1. Table Size - used to vary the size of an in-core table, thus allowing for modification of memory requirements.
2. Data Descriptions - modified by appropriate changes to respective PICTURE clauses. Used to vary computation accuracy requirements and processing time.

EXECUTE TIME PARAMETERS:

1. Constants - for random number generator
2. Processing Iterations - to vary CPU activity.
3. Accuracy Parameter - used to vary accuracy requirements.
4. Processing Deletion Switches - to indicate coding to be skipped.

NOTE: All parameters have default values -- see program listing for details.

Figure 6—Compute module parameters

complexity in the programs which would have rendered them completely unamenable to analysis.

(e) The design of each program (and of the set of programs as a whole) lends itself to extension, so that a wide range of task characteristics can be accommodated.

Each program is self-documented. A "prologue" is included for each and commenting is plentiful, though pertinent. External documentation consists of a "module overview" (see Figure 7), parameter specifications, experimental results, and a User Guide to assist an organization in implementing the programs and using the VP-Routine. We have avoided lengthy descriptions and detailed flowcharts because we question their usefulness.

PROJECT: SYNTHETIC BENCHMARKS

SEQUENTIAL MODULE OVERVIEW

PROGRAM-ID: SEQPRGRM

PURPOSE:

This synthetic program is designed to reflect the properties of a sequential file update process.

USAGE:

In its machine independent form SEQPRGRM is designed to be used in conjunction with the VP-routine (see references). In machine dependent form, SEQPRGRM is a stand-alone program.

ENVIRONMENT:

This program was developed on a UNIVAC-1100 System. It is designed to function correctly when translated by a COBOL compiler conforming to Federal COBOL standards as interpreted by the COBOL Compiler Validation System.

METHOD:

Master and detail sequential files are created, together with an in-core table. Timing for this program is then initiated. The master file is compared against the detail file until a key match is made. For each occurrence of a key match an update of the master file is made (creating a new master file), and a compute kernel is executed a varying number of times. When the detail file is exhausted, timing for this program is terminated and a summary record is written.

REFERENCES:

Navy COBOL Compiler Validation System User Guide
Information Systems Division (Op-91)
A synthetic job...Buckholtz, IBM Sys. J. (4), 1963

Figure 7—Example of a synthetic module overview

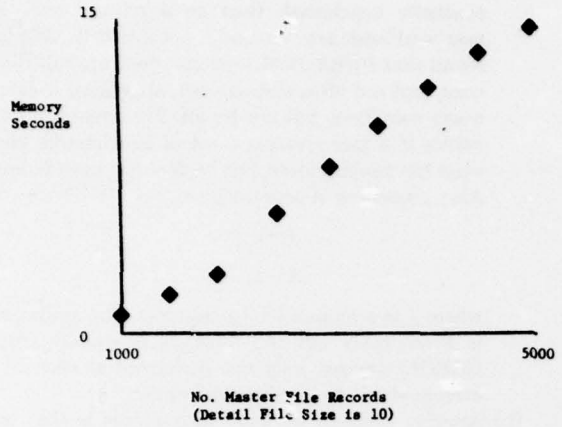


Figure 8—Sequential file update time as a function of master file size—no CPU activity, drum-resident files

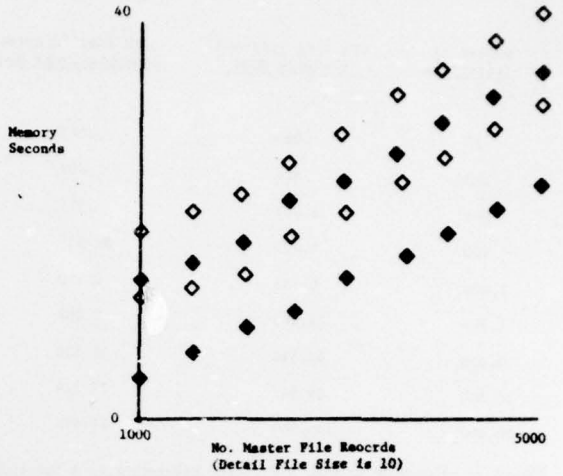


Figure 9—Sequential file update time as a function of master file size—no CPU activity, tape-resident files

The programs, documentation, and VP-Routine are collected on a 2400 foot magnetic tape reel. The User Guide and experimental results on program behavior are separately bound. The entire package is in the public domain.

Examples of processing results

A complete summary of processing results is beyond the scope of this paper, but we can discuss some of the more interesting of those results. All results mentioned are based on executions on a UNIVAC 1108 Unit Processor, under control of the EXEC-8 Operating System.

The "sequential I/O" module is the simplest of the file processing programs. Its function is to pass a master file against a detail file, creating a new master file. The files may reside on tape or direct access devices. A compute loop may be performed a variable number of times each time a master file record is updated. The processing includes a table search, and the size of the table is used to control memory requirements. All computations are self-checking. The program is similar in these and other characteristics to the PL/1 program described by Buchholz.[5]

Predictably, we found I/O time to be a linear function of master file size. This was true for FASTRAND (drum) resident as well as tape resident files. Repeated runs during different times of day showed that the curve reflecting the behavior of time as a function of master file size remained a straight line with constant slope, although the intercept value changed (Figure 8). In all these runs, only the master file size was varied (from 100 to 5000 records), with the detail file size fixed at 10 records), and only one pass through the compute loop was performed.

We processed a series of similar runs with all files residing on UNIVAC 8-C tapes. Again, running the program in a mix did not change the linear behavior of time as a function of file size (Figure 9). As before, the detail file size was held constant, and only one pass through the compute loop was performed on each record update. Thus, while other programs in a mix clearly affect the *quantitative* behavior of a sequential update task, they appear to have almost no effect on its *qualitative* behavior.

CPU time turned out to be a linear function of the number of repetitions through the compute loop.

Execution of the "compute" module produced some interesting results. The program generates a variable-sized table of uniformly distributed pseudo-random numbers, performs a "runs-up-and-down" test on them, and optionally produces printer output. A parameter controlling the number of processing iterations is used to vary the amount of CPU activity.

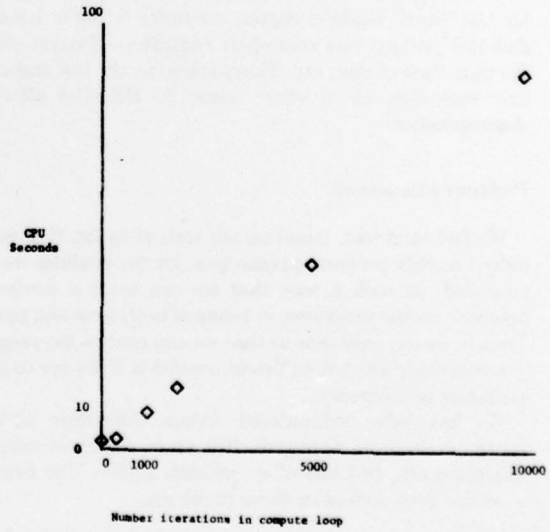


Figure 10—Compute module CPU utilization as a function of number of iterations in the computation loop

Number of Iterations	CPU Time (minutes) (Display Mode	CPU Time (minutes) (Computational Mode)
20	.083	.643
100	.816	.007
200	1.497	.515
500	4.525	1.906
1,000	9.531	2.990
1,700	14.945	5.062
5,000	45.524	14.156
10,000	89.941	23.324
20,000	158.507	47.696

Figure 11—Compute module CPU time utilization as a function of number of iterations in compute loop

When the number of iterations reached a certain threshold (usually 500) the CPU time varied linearly with this parameter. Below that point, however, we noticed some fluctuations (Figure 10). We believe this is due to the way the EXEC-8 dispatcher schedules jobs for CPU time. (It uses a variation of Corbato's time quantum charging algorithm.[9])

Figure 11 summarizes two executions, run under identical conditions. The only difference was that in one the usage of variables was "computational," in the other "display." As a program becomes CPU bound an exorbitant price is paid for the "machine independency" of data.

Figure 12 shows the relationship between memory time (for a given program, a memory second is defined as the occupation of 32K words of memory for a period of one second, during which time the program is undergoing either CPU or I/O activity) and the size of the file being sorted for the "sort" module. Again, we found a linear behavior, and this pattern was consistent regardless of other jobs in the mix, time of day, etc. Fluctuations at the low end of the line were due, as in other cases, to EXEC-8 allocation characteristics.

Problems encountered

We feel confident, based on our tests thus far, that we can indeed modify program parameters, for the modules we have produced, in such a way that we can force a predictable behavior on the programs, in terms of both time and pattern. This, however, only tells us that we can control the programs —a necessary but not sufficient condition if we are to create synthetic benchmarks.

We have also encountered certain difficulties with the synthetic program approach. Not all of these are unique to this approach, but this offers us little solace. The following were the most serious of these problems:

(a) Because synthetic programs tend to be stylized, they may produce surprising results. For example, an optimizing compiler can have a much greater impact on a synthetic benchmark than on a natural one. Yet, user workloads are "natural," not synthetic. We have found that PERFORM sections which are called only once, and not otherwise entered, are placed in-line by many compilers, but not by all. This creates no difficulties if a user creating a set of benchmarks knows what his compiler does, but he does not have to know. Also, sequences of code such as

$$I = I + 1$$

$$A = I,$$

where I is a loop-control parameter (the syntax here is FORTRAN but the principle is equally true of COBOL) are generally not performed as such by an even moderately intelligent compiler.

(b) Another problem we have encountered is that overwhelming side effects can occur in overly parameterized synthetic programs. For example, the COBOL PERFORM verb translates to 14 instructions on one system we executed under, while the MOVE verb translates to 1 instruction. Thus, using the PERFORM instruction to vary the number of times a MOVE instruction is executed leads to grossly misleading results when the PERFORM itself is the object of yet another PERFORM.

(c) One needs to understand the "native" system in some detail in order to develop benchmarks purporting to accurately reflect a given workload for that system. Some of the test results cited above, for example, were clearly due to the nature of the system on which the programs were executed. This means that guidelines on how to use the synthetic modules will differ with differing systems. Also, it is easy to create an unduly complex program (in terms of possible combinations of parameters) if the architecture of the native system is not understood. Repeating, for instance, a series of COBOL MOVE's, varying field sizes each time, accomplishes nothing more than what could be accomplished by moving a fixed size variable on IBM S/360 computers, since a single machine instruction, MVC (move character) is used regardless of field size. Yet, on a UNIVAC 1108, changes in object code

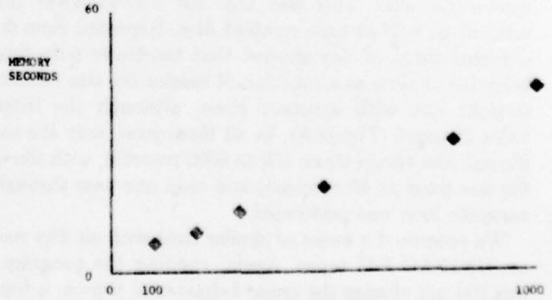


Figure 12—Sort module memory seconds utilization as a function of number of records sorted

do occur at certain field sizes. Also, moves of literals, numerics, and character fields are usually all performed in the same way, so that incorporating all of these in a program is simply adding to the combinations of parameters without really contributing to the value of the program.

(d) We see no evidence of a satisfactory way of modeling a workload. Even a simple I/O—CPU analysis of a file maintenance problem depends on a multitude of parameters: proportion of active to passive records, distribution and location of active records in the master file, number of instructions executed per active/inactive record, record size, frequencies with which instructions are executed, etc. This difficulty is seriously aggravated in a *mix* of programs. It is not at all clear that techniques for matching job parameters to mix parameters is feasible. The use of analytical models to characterize a job mix and thereby provide inputs to the synthetic programs[1] is clearly unsatisfactory, since the limiting factor would then become the analytical techniques themselves. This class of techniques is already regarded as grossly imprecise.

The use of software monitors for data collection is likewise unacceptable since they create serious instances of the "Hawthorne" effect.[10] This could possibly be compensated for, but with considerable difficulty.

In fact, it is important to note that *all* suggestions on how to model a workload rely on one of the evaluation techniques previously surveyed (*monitors, simulation, etc.*). Thus, we should not expect the synthetic mix approach to be an improvement over these.

The problem of "representativeness" which exists in natural benchmarks will simply not disappear just because we use synthetic programs. We have cited the system dependency of workload parameters (particularly as they apply to I/O time) and the sheer magnitude of the number of combinations of program parameter values. An equally crucial problem is the fact that the nature of a workload is time dependent. Any attempt to condense a workload into a, say, two-hour benchmark is bound to result in substantial homogenization, and some important characteristics could

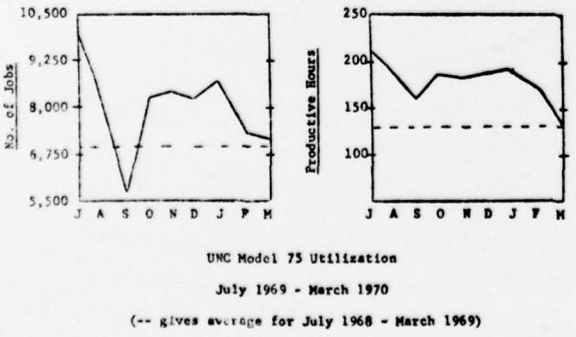


Figure 13—Monthly utilization profile (Source: Annual Report, University of North Carolina Computation Center, 1970)

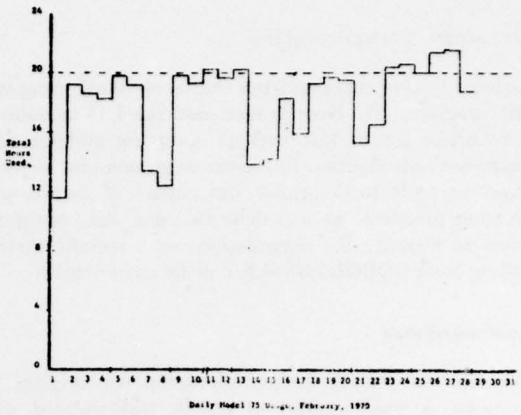


Figure 14—Daily utilization profile (Source: Annual Report, University of North Carolina Computation Center, 1970)

be lost. As a simple example, the annual workload of a computer center, in terms of productive hours, is given in Figure 13. It suggests that there is plenty of excess capacity. Yet the workload on a typical mid-week day shown in Figure 14 indicates that for this period the system was saturated. We know of no satisfactory techniques which allow us to model this behavior for the purpose of building benchmarks.

CONCLUSIONS

Can a controllable job mix be constructed?

We believe, on the basis of our experience thus far, that task-oriented synthetic programs can be combined into a mix which can be controlled to exhibit desired processing time, memory, I/O time, and I/O devices utilization characteristics. There have been other efforts that bear this out.[11] We plan additional testing on a variety of systems so as to learn more about some of the system dependencies we have encountered.

Can a workload be profiled?

We do not believe that it is possible to arrive at a generalized, comprehensive, and accurate model of system workloads except in the most trivial cases. We can certainly retrofit. That is, we can accept a workload definition based on the synthetic program parameters. We also believe that this need not impede the use of synthetic programs in benchmarks. In this, we strongly support the view expressed by J. C. Strauss. In a recent paper[12] on the use of natural benchmarks, he stated that, based in part on prior experience and on the difficulties encountered, "it was felt more important that the behavior of the benchmarks be well understood and cover a broad range of important system features than that the complete benchmark series be representative of the general workload."

Other uses for synthetic programs

Isolated system characteristics can be exercised using synthetic programs. We have in fact used the I/O modules in our reference set to test various operating systems data management capabilities. Synthetic programs also serve as convenient tools to determine the impact of certain programming practices, as was done in using the "compute" module to measure the degradation, on a specific system, resulting from COBOL DISPLAY mode computation.

A recommendation

We feel our testing has substantiated our original assumptions. A small number of simple, task-oriented, synthetic programs can be combined into a fairly rich and versatile job mix. A relatively small number of parameters is sufficient to enable a single program to reflect the characteristics of a broad class of applications. Also, individual modules have proven useful in exercising isolated computer system features, such as I/O handling. Finally, if one accepts a "modest" workload characterization, aimed more at reflecting extremities and crucial areas rather than comprehensiveness, it is possible and reasonable to construct a benchmark from a set of synthetic modules.

Synthetic programs are neither difficult nor expensive to produce. Our present set, admittedly small, was designed, coded, and debugged in two calendar months. An additional three months were required for experimentation, packaging, and system documentation. These times do not consider the VP-Routine, which was already available. Total manpower used for the effort amounted to four man-months. Total cost, including machine time, clerical support, and salaries was under $6,000. Furthermore, the system is available to anyone upon request. Thus, we feel we have made a small investment for a product which has already given a substantial payoff, in what we have learned if nothing else.

A reference set of "controllable" programs is a useful tool for any data processing installation. Our concern was primarily with benchmarks for system selection. We have indicated that performance measurement is a related area of application. System sizing, throughput estimates against a changing workload, expected response time to a varying stimulus, and availability measurements are other reasonable applications for a set of synthetic modules. The modesty of the effort required to produce such a set certainly commends further study.

REFERENCES

1. Lucas, H. C., "Performance Evaluation and Monitoring," *ACM Computing Surveys*, 3, 3, 1971.
2. Dopping, D., "Test Problems Used For the Evaluation of Computers," *Bit*, 2, 4, 1962.
3. Gosden, J. A. and R. L. Sisson, "Standardized Comparisons of Computer Performance," *Proc. 1962 IFIP Congress.*
4. Joslin, E. O., "Application Benchmarks: The Key to Meaningful Computer Evaluations," *Proc. 20th ACM Nat. Conf.*, pp. 27-37, 1965.
5. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM System Journal*, Vol. 8, No. 9, 1969.
6. Byrne, T. A., et al., "A Job Selection Simulation Model," *Symposium on the Simulation of Computer Systems* (ACM), June, 1973.
7. Hesser, W. A., "Creation of a Simulation Model From Hardware Monitor Data Using the SAM Language," *Symposium on the Simulation of Computer Systems* (AMC), June 1973.
8. Baird, G. N., "The DOD COBOL Compiler Validation System," *Proc. FJCC*, 1972.
9. *UNIVAC 1100 Series Operating System Programmer Reference*, UP-4144, Sperry-UNIVAC (1973).
10. Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement," *IEEE Computer Journal*, July/August, 1972.
11. Wood, David C. and Ernest H. Forman, "Throughput Measurement Using a Synthetic Job Stream," *Proc. 1971 FJCC*, AFIPS Press, Vol. 39.
12. Strauss, J. C., "A Benchmark Study," *Proc. 1972 FJCC*, AFIPS Press, Vol. 41, Part II.

BIBLIOGRAPHIC DATA SHEET	1. Report No. FCCTS/TR-77/07	2.	3. Recipient's Accession No.
4. Title and Subtitle An Experiment in the Use of Synthetic Programs for System Benchmarking			5. Report Date 9 May 1977
			6.
7. Author(s) Paul Oliver, George Baird, Margaret Cook, Arnold Johnson and Patrick Hoyt			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Software Development Division ADPE Selection Office Department of the Navy Washington, D. C. 20376			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address ADPE Selection Office Department of the Navy Washington, D. C. 20376			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

The Federal COBOL Compiler Testing Service has experimented with the use of synthetic programs for system benchmarking. The results of this experiment are discussed here.

17. Key Words and Document Analysis. 17a. Descriptors

COBOL
Benchmarking
Performance Evaluation
Synthetic Programs

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group 09/02

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited.	UNCLASSIFIED	8
	20. Security Class (This Page) UNCLASSIFIED	22. Price